ISBN: 978-972-8939-47-2 © 2011 IADIS

A MODEL-BASED MONITORING ARCHITECTURE FOR HETEROGENEOUS ENTERPRISE SERVICES AND INFORMATION SYSTEMS

Félix Cuadrado, Rodrigo García-Carmona, Juan C. Dueñas and Álvaro Navas
Departamento de Ingeniería de Sistemas Telemáticos, ETSI Telecomunicación,
Universidad Politécnica de Madrid, Spain

ABSTRACT

Runtime management of distributed information systems is a complex and costly activity. One of the main challenges that must be addressed is obtaining a complete and updated view of all the managed runtime resources. This article presents a monitoring architecture for heterogeneous and distributed information systems. It is composed of two elements: an information model and an agent infrastructure. The model negates the complexity and variability of these systems and enables the abstraction over non-relevant details. The infrastructure uses this information model to monitor and manage the modeled environment, performing and detecting changes in execution time. The agents infrastructure is further detailed and its components and the relationships between them are explained. Moreover, the proposal is validated through a set of agents that instrument the JEE Glassfish application server, paying special attention to support distributed configuration scenarios.

KEYWORD

Service-Oriented Architectures, Heterogeneous Systems, Distributed Systems, Information Systems, Model-Driven Engineering.

1. INTRODUCTION

The actual economic climate and globalization have forced companies to face an ever increasing competition. This situation forces them to develop new products in an agile way and quickly satisfy the demands of its consumers. The most usual way to reach these targets is by adopting the service oriented paradigm, which promotes the division of work in small components that work at unison. To enable this, a robust support infrastructure is mandatory.

Concerning enterprise applications, quality of service is one of the most important factors that need to be taken into account. Quality of service is built upon small response times and a high availability, and these non-functional requirements are partially covered by the execution infrastructure. This infrastructure is composed of several heterogeneous servers with specific roles, distributed over the network and grouped in clusters. However, this kind of system greatly complicates the management processes [12], such as status diagnosis, change evaluations or execution of operations intended to improve performance.

These tasks are usually executed by a manual operator. But this has a huge cost and does not provide the agility needed for the processes. This way, to reach the theoretical advantages of a service oriented architecture, a higher grade of automation in management operations is needed. But this is not an easy task, since the composition of the management environment tends to be unknown during design phase (amount of elements and system types, such as databases, application servers or business rules). This fact adds a huge amount of complexity to the management process.

The solution to this problem lies in the adoption of a model-based abstraction layer. This layer would allow the definition of management operations and environment elements in execution time. With it, the resulting system can be more easily tailored to the specific environment details.

To enable this, a set of intermediation agents that hide the complexities of the environment to the management system have to be developed. These agents must both translate the details of the environment to the abstract model and the generic operations to commands specific to the interface of each element.

In this paper we present an information model and an architecture which together fulfill the need to obtain the execution time information of a complex information system, and perform deployment and configuration changes over it. The proposed solution automatically adapts to the specific characteristics of each environment, combining an extensible architecture with model abstractions to represent the environment state and the management operations. The solution has been designed keeping in mind the specific requirements of enterprise banking systems. This work has been developed under the CENIT ITECBAN project, which aims to create a complete core banking solution that leverages the recent paradigms of model-driven engineering and service oriented architecture.

The article is structured as follows: In the next section we summarize the most relevant initiatives from both the industry and academia concerning the deployment and configuration change execution over distributed environments. In the third section we explain why none of the solutions introduced in the state of the art fulfill all the requirements. We also detail our information model here. Following this, in the next section we describe the most relevant aspects of the proposed architecture, explaining each component in detail. In the fifth section we validate our approach using some of the agents developed: the ones designed to work with the open source Glassfish application server. Lastly, we finish this article with the conclusions we extract from our work. At the same time we propose several research topics that we think our work could lead to in the future.

2. STATE OF THE ART

2.1 Management Service Information Models

Model-Driven Engineering (MDE) is a methodology based on abstracting real elements as models. Models only contain the relevant information from the application domain, hiding the underlying complexity derived from unneeded characteristics. Metamodels govern the semantics, characteristics and restrictions of models, avoiding potential ambiguity. These characteristics are well known in the domain of network and distributed management, where management information models are the prime mechanism for capturing a complete representation of the managed system. Consequently, there are several standardization initiatives for representing the relevant management information. We briefly present the most important initiatives here.

CIM (Common Information Model) [8] is an information model that describes the information of an enterprise distributed system. The standard is maintained by an enterprise consortium, the DMTF (Distributed Management Task Force). CIM provides an object-oriented model for representing the managed elements. The specification extensively uses inheritance mechanisms to provide a modular and extensible characterization of them. The modular structure of CIM is composed by a common core that defines the basic elements, and a set of extensions. Each extension provides additional details over an aspect of the system (e.g., databases, application servers, policy definition or hardware devices). Extensions can either extend the base model, or further refine the concepts presented by other extensions. On the one hand, this approach provides a rich characterization of every managed element, with a specific class governing its attributes and management operations. On the other hand, the standard is so extensive that supporting tools only implement specific subsets of the implementation, and the required modeling effort for covering additional elements complicates its adoption to very heterogeneous systems.

The Object Management Group (OMG) also defines a distributed managed information model: the D&C (Deployment & Configuration of Component-based Distributed Applications) specification [10]. This standard also uses object orientation to model the managed elements, but follows a different approach. Instead of attempting to define a class for each different managed element, D&C opts for a simple and flexible approach. Every managed element is a resource, modeling an entity with an identifying name, classified into one or more types. Resources model every element, ranging from physical artifacts, computing nodes, network routers, as well as the software applications and services running on top of them. The management information of each resource is provided through a set of properties (name-value pairs). The

combination of the type mechanism and characterization through properties allows management systems to operate heterogeneous elements using the same abstractions. This approach is better suited for those types of environments, but the specification is still limited in some aspects. D&C lacks support for representing software and services, as key elements such as deployable units, version management, dependencies or bindings among logical components cannot be represented with those concepts.

Finally, we present the MUWS (Management Using Web Services) specification promoted by OASIS [2]. This specification provides an information model and management protocol to manage distributed resources using Web Services. Managed elements are modeled also as resources. Resources offer a set of management capabilities that can be invoked by the management infrastructure. Basic capabilities include description (providing resource name and version), state, metrics reporting or configuration (using a property name-value system). This mechanism is designed with extensibility in mind, so that additional capabilities can be developed for specific resources. In its simplicity, this approach is similar to the one followed by D&C, but it provides a fundamental concept for successfully managing software elements in a rapidly evolving world: the concept of resource versioning.

The three analyzed information modeling standards share a common foundation, which is an object-oriented approach, and the concept of resources as the base management elements. However, no specification can be directly applied to automate distributed heterogeneous service management. CIM approach is too restrictive for heterogeneous systems, as the effort to model each different element greatly complicates the development of autonomous elements. The generic approach of the other two initiatives is better suited for that specific requirement. D&C provides the key concept of types, but the standard lacks adequate support for characterizing the software elements. Finally, MUWS is the most basic of the three specifications, as it only presents the base resource concept, and leaves additional characterization for capability extensions.

2.2 Heterogeneous Systems Management

Traditional system management processes are governed by a human operator that interacts with the system through an administration console. However, the ever growing complexity of the managed systems, the degree of heterogeneity, and the distributed nature of these elements greatly complicate its application. There is a clear need for tools and frameworks that can (at least partially) automate these management tasks. Instead of simply presenting all the gathered information, tools must aggregate all the management data, filter out the non-relevant aspects, evaluate the current state, decide if any corrective action is required and finally invoke its execution. This idea of self-managed systems has been promoted since 2003 under the autonomous computing paradigm [7]. The cost of operation of a self-managed system is considerably reduced, as it automatically carries out management activities without the need of human intervention. Each autonomic manager can operate independently or coordinate with other agents, as well as work under the supervision of human administrators. This way, this paradigm presents a general high-level goal, instead of an approach that is completely incompatible with existing processes and infrastructure.

PBM (Policy-Based Management) [11] proposes to implement management automation through the use of policies. A policy formalizes a guideline or criteria that must be met by the managed systems. Policies are usually represented as rules, allowing management systems to automatically reason about the collected management information, and invoke operations to correct how the managed system works. Policies are a useful tool for implementing autonomic behavior.

There are several examples in the literature on how an autonomic manager can be implemented using policies. The PMAC (Policy Management for Autonomic Computing) platform [9] combines policies expressed in the ACEL language in order to provide an autonomic network manager. Focale [1] adopts a similar approach, implementing an autonomic network manager through the use of ontologies. However, none of them addresses the problem of automating the management of an enterprise, distributed heterogeneous information system. Nonetheless, these approaches can also be applied to this domain. The only requirement for the adoption of policies or similar mechanisms to automate management is the existence of an information model that captures with sufficient detail the relevant information about the runtime state, operation constraints, and potential operations. The following section presents our proposed information model.

3. RESOURCE INFORMATION MODEL

We have defined an information model that builds on top of the common ground of the analyzed standards, and at the same time provides the required concepts and capabilities for automated management. The model abstracts both the physical (or virtualized) runtime infrastructure and the software applications and services that execute on top of it. Figure 1 shows the main elements of the runtime part.

Resources are the main model elements; they represent every manageable element of the system. Resources are characterized by name, version and a set of properties. Additionally, every resource belongs to a type, which intrinsically classifies similar elements. This way, it is possible to define restrictions that can be matched by any element belonging to a type, simplifying the definition of automated policies and algorithms. Automated reasoning is achieved as the model definition is complemented with a type taxonomy that provides agreed types for the main elements in banking services and information systems (ranging from hardware elements to software middle ware and services).

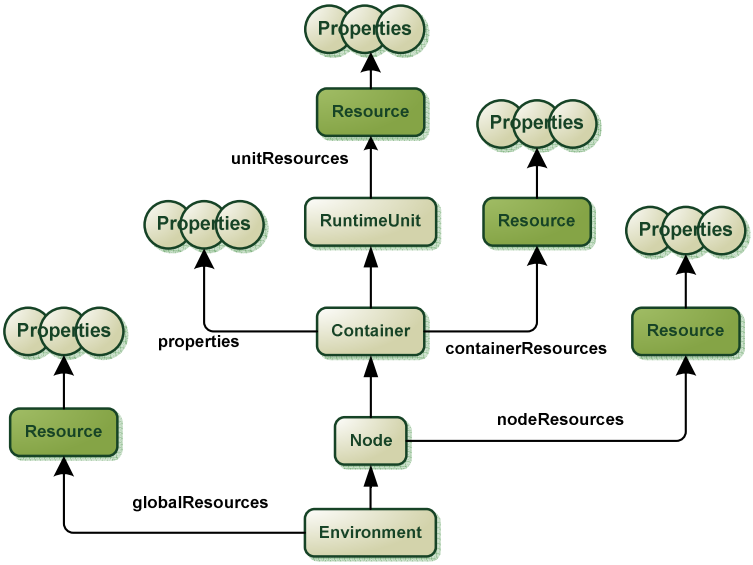

Figure 1. Runtime System Model

The *runtime metamodel* is composed by a set of resource subclasses that characterize the main elements that constitute the management system. The subclasses constitute a hierarchy that mirrors the topology of the environment. The root element is the Environment, whose main elements are identical to the D&C Deployment Target element. The Environment contains system-wide Resources (e.g. a DNS server), as well as the physical runtime topology, through a list of Interconnect and Bridge elements. Finally, an Environment aggregates the runtime computing resources as a set of Node elements. Nodes are Resource subclasses that comprise the hardware, firmware and operating system of managed elements, with specific capabilities such as memory, RAM, or available TCP ports modeled as Node Resources. Nodes also host a set of Containers, which represent the application servers, DBMS (Database Management Systems), and BRMS (Business Rule Management Systems) that host the software artifacts that provide the specific functionality. In contrast to D&C they are specifically modeled as they play a fundamental part in services deployment and configuration. Containers host the Runtime Units, which represent the software artifacts that are deployed over the environment. Units provide at execution time services, libraries and functional components that are also modeled as resources (with the type identifying its specific characteristics).

The *software metamodel* describes the software elements before they are deployed at the runtime environment. The model characterizes the Deployment Units, and provides information for their automated management such as the logical dependencies with other components, or the runtime restrictions for their correct execution (e.g. demanding a minimum amount of RAM, or the exclusive access to a specific TCP port). Both models share the common concepts of Resource and Unit. This way the restrictions defined by the software model can be automatically checked at the runtime model.

4. MANAGEMENT AGENT ARCHITECTURE

The modeling abstractions presented at the previous section capture all the relevant information for managing a heterogeneous distributed infrastructure. However, they must be supported by an instrumentation infrastructure. Instrumentation is performed by a set of agents that mediate between the management system and the managed elements. These agents must perform two vital functions. First, they must be able to retrieve the information about the current state of the managed elements, and represents it with the presented model abstractions. Additionally, agents must be able to apply deployment and configuration operations to the runtime environment, by an invocation of the different management interfaces.

The runtime infrastructure is an evolving ecosystem, with continuously changing software and hardware. Additionally, there is a great degree of diversity between the topology and selected technologies for the different solutions that are adopted by different organizations and departments. Because of those factors, the agents infrastructure should support an additional requirement: it must detect changes in the infrastructure, automatically aggregate the information from new elements and alert about missing parts.

We have fulfilled these requirements in by adopting a layered design. Figure 2 shows the main elements of the instrumentation architecture. Agents operate at different abstraction levels so that they can automatically aggregate the management information and can seamlessly communicate with the specific managed elements. We now present the main elements of the architecture.

Context Gatherer: These components monitor a runtime node or container; they collect information about its current state and changes by connecting to the specific management APIs and convert the retrieved data to resources from the model. Examples of the types of information that is retrieved by these components include operating system details (name, version, libraries), hardware resources (processor, available memory, devices) or container configuration (http port, deployed units). Each physical runtime element is instrumented by a specific Context Gatherer, whose information is integrated through the Adaptor design pattern.

Actuator: Invokes deployment and configuration activities at the runtime containers. Example operations include configuring a container or installing a deployment unit. Each Actuator is registered to operate specific types of containers, which allows automatic matching between generic change orders and specific execution.

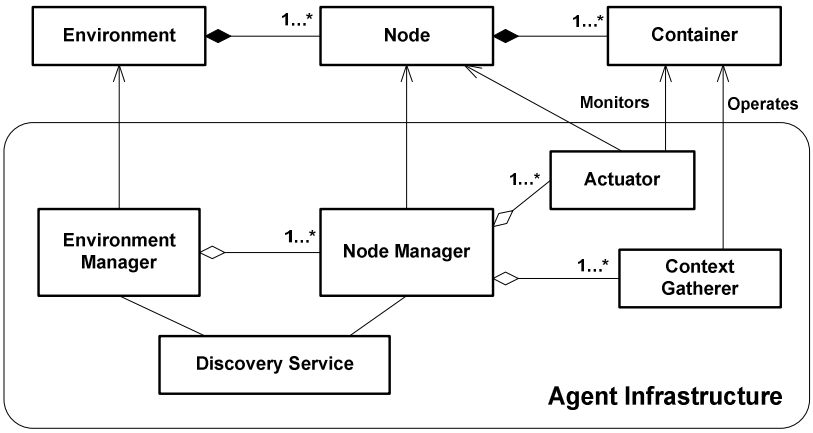

Figure 2. Agents Instrumentation Infrastructure

Node Manager: Manages the available information and invoked changes at a runtime node. This component aggregates both *Context Gatherers* and *Actuators* registered to one node. Node Managers are not specific to each managed node, they abstract from the specific details. For monitoring purposes, they aggregate the information collected by the specific Context Gatherers, already converted to the model.

Environment Manager: This component is the root of the agents infrastructure, coordinating all the management activities that take place at the runtime environment. The Environment Manager controls the deployed Node Managers, constructing a complete model of the runtime state and coordinating the execution of change plans. Communications between these agents is performed through an asynchronous event mechanism, in order to reduce network load of the instrumentation layer.

Discovery Service: The Environment Manager must be aware of all the existing Node Manager agents. The designed architecture allows either manual configuration of the instance locations or automatic agent discovery. The latter mechanism is implemented through the use of a service discovery protocol based in DNS-SD [3]. This way, appearing Node Managers are detected by the Environment Manager and are added to the agents network automatically. Similarly, agent and node malfunctions are also detected and notified to the Environment Manager, reporting a runtime incidence.

5. CASE STUDY: GLASSFISH JEE INSTRUMENTATION

The ITECBAN project has developed a service-based, complete banking solution. The functionality is provided by application servers, business rule systems, business processes and information systems. In addition to the infrastructure, the project has created a complete set of supporting middleware and services that enable the provisioning and operation of the developed services. Distributed management is one of the main functions provided, using the agents infrastructure to instrument the runtime environment. The direct interaction with the physical systems is performed with Context Gatherer and Actuator agents, while the global view of the system is coordinated with Node Managers and a central Environment Manager element.

In order to clarify how the presented information model can successfully capture the information from elements with such different characteristics we present the technical details behind one of the developed agents: the Glassfish Context Gatherer. Glassfish is an open source project led by Sun/Oracle that provides the reference implementation for the Java Enterprise Edition standard. The presented agent connects to Glassfish management interfaces and obtains from them a Container instance of the information model.

Application servers that meet the Java EE standard (such as Glassfish) must implement two management specifications. These specifications are the JSR (Java Specification Request) 77 [6] and the JSR 88 [4], that respectively define monitoring and deployment interfaces based upon JMX management beans. However, the first one does not provide a rich enough description of the runtime state, leaving to each application server implementation the responsibility to fill this gap. Glassfish achieves this with the AMX (Advanced Management eXtensions) [5] specification, that extends the management beans with the required methods. Consequently, in order to retrieve the server state, the Glassfish Context Gatherer must use JMX to connect to the MBeans Server and interact with the AMX and JSR 77 interfaces.

In order to cope with the heavy throughput and reliability requirements of an enterprise domain, Glassfish servers are deployed in a cluster configuration. This way, there is a pool of server instances with replicated functionality, and a load balancer that routes the requests among them. Glassfish agents monitor that instances are correctly running, and a cluster definition defines what resources are shared among all the instances (such as connections to remote systems, or datasources to access the information systems), as well as what software components should be deployed at the cluster. Figure 3 shows a graphical representation of the decided mapping. The left-hand side shows the available information from AMX objects, while the right-hand side shows the equivalent information model elements. In the following paragraphs we present how we have modeled all those elements with the presented abstractions.

A cluster configuration expands over multiple computing nodes, distributing the cluster definition, the different instances and the load balancer. However, for management purposes all those elements constitute a single entity, which corresponds to a Container in our model. Cluster shared resources (e.g. datasources), as well as deployed software artifacts, should be modeled as container resources.

Additionally, the management system must be aware of the pool of cluster instances. Each one of them will be modeled as a resource of the container, belonging to the type *j2ee.cluster.instance*. Instance-specific characteristics, such as the http service port, will be modeled as properties. Finally, the information about the load balancer that manages requests to the cluster will be represented as another container resource, of type *j2ee.cluster.lb.*

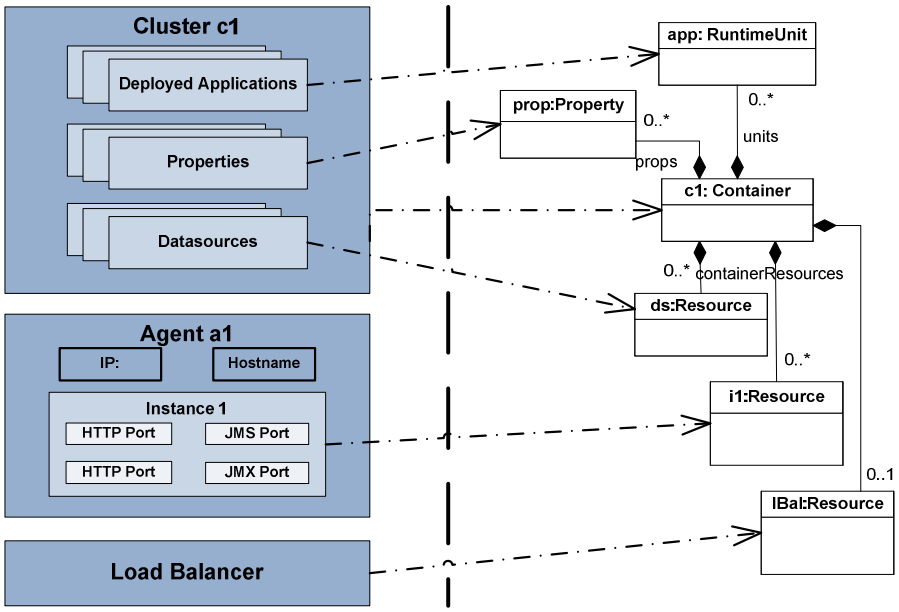

Figure 3. Mapping of AMX Managed Beans to the Generic Information Model

This way, the developed Context Gatherer will retrieve the information in the container specific format and will build model instances by applying the defined mapping. Additionally, Actuators will translate generic management operations and invoke the specific management interfaces. The following code listing presents an XML fragment of the collected information from a Glassfish cluster, modeled as a Container element of our information model.

```
<Container>
   <name>cluster1</name>
   <containerTypes>
      <containerType>
         <name>es.itecban.deployment.container.jee.glassfish.cluster</name>
         <version>2.1</version>
     <containerResources>
      <containerResource>
       <name>instance4</name>
       <runtimeResourceTypes>
          <type>es.itecban.deployment.j2ee.cluster.instance</type>
       </runtimeResourceTypes>
       <runtimeResourceProperties>
        <resourceProperty>
          <name>HostName</name>
          <value>segovia.dit.upm.es</value>
        </resourceProperty>
       </runtimeResourceProperties>
      </containerResource>
     </containerResources>
    </nodeContainer>
    <nodeContainer>
     <name>cluster2</name>
     <containerResources>
      <containerResource>
       <name>instance2</name>
        <resourceProperty>
          <name>HostName</name>
          <value>indra1.dit.upm.es</value>
        </resourceProperty>
      </containerResource>
      <containerResource>
       <name>instance3</name>
        <resourceProperty>
          <name>HostName</name>
          <value>indra3.dit.upm.es</value>
        </resourceProperty>
      </containerResource>
</Container>
```

6. CONCLUSIONS

In this paper we have presented a complete model and agent architecture to instrument distributed heterogeneous information systems, enabling its automated management. The system seamlessly interfaces between the generic models and the heterogeneous elements of the managed system, thanks to an adaptation layer provided by the combination of the information model and the extensible agents infrastructure. Model abstraction enables the management system to reason generically about the environment, being oblivious to the specific implementation details. Context Gatherers are the architecture elements that are specific to the environment characteristics. Additionally, the proposed architecture can respond to changes in the runtime environment, such as the appearance or disappearance of a node.

We have successfully applied this proposal to manage a distributed banking infrastructure. We present in this article the fundamental concepts from one of the main Context Gatherers, the responsible of the instrumentation of Glassfish Application servers. The selected component is interesting because of its importance as the reference implementation of the JEE specification, and also because of the challenge of modeling application server clusters.

The work presented offers many opportunities for future developments, as the definition of management policies for modeled environments, or the implementation of an automatic feedback loop that would not only detect the appearance or disappearance of a node in the system, but also plan a modification of its state to take advantage of this event.

ACKNOWLEDGEMENTS

The work presented in this article was developed in the context of ITECBAN project, partially funded by CDTI (Centro para el Desarrollo Tecnológico Industrial) and MITYC (Ministerio de Industria, Comercio y Turismo de España).

REFERENCES

[1] Agrawal, D., Lee, K. and Lobo, J., “Policy-Based Management of Networked Computing Systems”, IEEE Communications Oct 2005.

[2] Bullard, V., Vambenepe, W., WSDM, Web services distributed management: Management using Web services (MUWS 1.1), OASIS Standard, August 2006. Available at http://www.oasis-open.org/committees/download.php/20576/wsdm-muws1-1.1-spec-os-01.pdf

[3] Chesire, S., Krochmal, M., DNS-SD, DNS Service Discovery, IETF Internet Draft, available at: http://files.dns-sd.org/draft-cheshire-dnsext-dns-sd.

[4] Dochez, J. et al, “JSR 88, Java EE Application Deployment”, available at http://jcp.org/en/jsr/detail?id=88

[5] Glassfish Development Team, “AMX Specification”, available at https://glassfish.dev.java.net/javaee5/amx

[6] Hrasna, H. et al, “JSR 77, J2EE Management”, available at http://jcp.org/en/jsr/detail?id=77

[7] IBM, An architectural blueprint for autonomic computing, IBM Whitepaper, available at http://www-03.ibm.com/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf

[8] Information Model (CIM) specification v2.1, DMTF standard, http://www.dmtf.org/standards/cim

[9] Jennings, B., van der Meer, S., Balasubramaniam, S., Botvich, D., Foghlú, M.O., Donnelly, W. and Strassner, J., “Towards Autonomic Management of Communications Networks”, IEEE Communications, Oct 2007

[10] Object Management Group, “Deployment and Configuration of Component-based Distributed Applications Specification.” version 4.0 formal/06-04-02, 2006.

[11] Verma, D.C., Simplifying Network Administration Using Policy-Based Management, IEEE Network, March 2002

[12] Versendaal, J., Gils, B.V. Janssen, W. “Operational Excellent Application Management: A Case Study at an Insurance Company” IADIS International Conference Information Systems 2010, pp.191-198, 2010. IADIS IS 10